DRAFT NIST Special Publication 800-132

Recommendation for Password-Based Key Derivation

Part 1: Storage Applications

Meltem Sönmez Turan, Elaine Barker, William Burr, and Lily Chen

Computer Security Division
Information Technology Laboratory

C O M P U T E R   S E C U R I T Y

June 2010

U.S. Department of Commerce
Gary Locke, Secretary

National Institute of Standards and Technology
Patrick D. Gallagher, Director

Abstract

This Recommendation specifies techniques for the derivation of master keys from passwords to protect electronic data in a storage environment.

KEY WORDS: Password-Based Key Derivation Functions, Salt, Iteration Count, Disk Encryption

Acknowledgements

The authors, Meltem Sönmez Turan, Elaine Barker, William Burr, and Lily Chen of the National Institute of Standards and Technology (NIST), would like to thank their colleagues, Shu-jen H. Chang, Morris Dworkin, Allen Roginsky, John Kelsey of NIST and Kevin Igoe and Rich Davis of the National Security Agency for helpful discussions and valuable comments.

Table of Contents

Recommendation for Password Based Key Derivation

1 Introduction

The randomness of cryptographic keys is essential for the security of cryptographic applications. However, in some applications, such as the protection of electronically stored data, passwords are the only input required from the users who are eligible to access the data. Due to the low entropy and possibly poor randomness of those passwords, they are not suitable to be directly used as cryptographic keys.

This Recommendation specifies a family of Password-Based Key Derivation Functions (PBKDFs) for deriving cryptographic keys from passwords for the protection of electronically-stored data.

2 Authority

This publication has been developed by the National Institute of Standards and Technology (NIST) in furtherance of its statutory responsibilities under the Federal Information Security Management Act (FISMA) of 2002, Public Law 107-347. NIST is responsible for developing standards and guidelines, including minimum requirements for federal information systems, but such standards and guidelines shall not apply to national security systems without the express approval of appropriate federal officials exercising policy authority over such systems. This guideline is consistent with the requirements of the Office of Management and Budget (OMB) Circular A-130, Clause 8b(3), Securing Agency Information Systems, as analyzed in Circular A-130, Appendix IV: Analysis of Key Clauses. Supplemental information is provided in A-130, Appendix III.

Nothing in this publication should be taken to contradict the standards and guidelines made mandatory and binding on federal agencies by the Secretary of Commerce under statutory authority. Nor should these guidelines be interpreted as altering or superseding the existing authorities of the Secretary of Commerce, Director of the OMB, or any other federal official. This publication may be used by nongovernmental organizations on a voluntary basis and is not subject to copyright in the United States. Attribution would, however, be appreciated by NIST.

3 Definitions, Acronyms and Symbols

3.1 Definitions

Approved	FIPS-approved and/or NIST-recommended. An algorithm or technique that is 1) specified in a FIPS or NIST Recommendation; or 2) adopted in a FIPS or NIST Recommendation; or 3) specified in a list of NIST-approved security functions.
Authenticated encryption	A function in which plaintext is encrypted into ciphertext, and a MAC is generated on the plaintext or ciphertext and, optionally, on associated data that is not encrypted.
Cryptographic algorithm	A well-defined computational procedure that takes variable inputs that may include a cryptographic key, and produces an output.
Cryptographic key (key)	A binary string that is used as a parameter by a cryptographic algorithm.
Data protection key	A key or a set of keys used to protect or recover data, verify the authenticity or integrity of the protected data or to protect the private key to generate digital signatures.
Decryption	The process of transforming ciphertext into plaintext using a cryptographic algorithm and key.
Digest size	The output length of a hash function.
Encryption	The process of transforming plaintext into ciphertext using a cryptographic algorithm and key.
Entropy	A measure of the amount of uncertainty in an unknown value.
Iteration count	The number of times that the PRF is called to generate a block of keying material.
Key	See cryptographic key.
Key derivation	The process of deriving keying material from a key or password.

Key generation	The process of generating keys for cryptographic algorithms.
Keying material	A binary string that can be parsed into one or more non-overlapping segments of appropriate lengths to use as symmetric cryptographic keys.
Master key	In this Recommendation, a master key is the keying material that is output from an execution of the PBKDF.
Message (Data) authentication	A mechanism to provide assurance of the origin and integrity of a message.
Message authentication code (MAC)	A cryptographic checksum that is generated on data using a cryptographic algorithm that is parameterized by a symmetric key. The MAC is designed to provide data origin authentication and detect both accidental errors and the intentional modification of the data. Also called an authentication tag.
Password	A character string known only by a specific entity (e.g. a user) that is used to authenticate the identity of a computer system user and/or to authorize access to system resources. In this Recommendation, a password is used to derive keying material.
Protected data	In this Recommendation, protected data implies that the data is encrypted, authenticated, or both encrypted and authenticated.
Pseudorandom Function	A function that can be used to generate output from a random seed and a data variable, such that the output is computationally indistinguishable from truly random output.
Recover data	To decrypt ciphertext data and/or to verify the authenticity of data.
Salt	A non-secret binary value that is used as an input to the key derivation function PBKDF specified in this Recommendation to allow the generation of a large set of keys for a given password.

3.2 Acronyms

Acronyms	**Meaning**
DPK	Data Protection Key
GCM	Galois Counter Mode
HMAC	Keyed-Hash Message Authentication Code (as specified in FIPS 198-1 [1]).
KDF	Key Derivation Function
MK	Master Key

PRF	Pseudorandom Function
PBKDF	Password-based Key Derivation Function
SHA	Secure Hash Algorithm

3.3 Symbols

Symbol	**Meaning**
$\oplus$	Bit-wise exclusive-or.
$\|\|$	Concatenation.
$\lceil a \rceil$	The ceiling of *a*: the smallest integer that is greater than or equal to *a*. For example, $\lceil 5 \rceil = 5$, $\lceil 5.3 \rceil = 6$, and $\lceil -2.1 \rceil = -2$.
C	Iteration count.
hLen	Length of output of PRF in bits.
Int(*i*)	32-bit encoding of integer *i* with the most significant bit on the left.
kLen	Length of output of PBKDF in bits.
len	An integer that represents the number of output blocks to be concatenated in order to obtain *kLen* bits of master key.
mk	Master key derived using the PBKDF.
P	Password, represented as a binary string.
purpose	The part of the salt that may be application, message or user-specific.
rv	The randomly generated part of the salt.
S	Salt, represented as a binary string.
sLen	Length of the salt in bits.
T<0, 1, …, r-1>	Truncation of the binary string *T* that retains its first *r* bits.

4 General Discussion

This Recommendation specifies a family of functions to derive cryptographic keying material from a password. The derived keying material is called a Master Key (MK), denoted as *mk*. MK is used either to form one or more Data Protection Keys (DPKs) to protect data through segmentation or derivation using an **approved** key derivation function (KDF) as defined in [2], or to protect DPKs. If the MK is used to protect DPKs, the protection **shall** use an **approved** authenticated encryption mode, such as defined in [3-5], or an **approved** key protection method.

5 Password-Based Key Derivation Functions

A password is a string of characters that is usually chosen by users. In most applications, passwords are used to authenticate a user in order to gain access to a resource. Since most user-chosen passwords have low entropy and weak randomness properties as discussed in Appendix A.1, these passwords **shall not** be used directly as cryptographic keys. However, in certain applications, such as protecting data in storage devices, the password is the only secret information that is available to the cryptographic algorithm that protects the data. This section specifies a family of password-based key derivation functions (PBKDFs).

KDFs are deterministic algorithms that are used to derive cryptographic keying material from a secret value, such as a password. Each PBKDF in the family is defined by a pseudorandom function (PRF), and a fixed iteration count, denoted as *C*. The input to an execution of PBKDF includes a password, denoted as *P*, a salt, denoted as *S*, and an indication of the desired length of the key in bits, denoted as *kLen*. Symbolically:

$$mk = \mathrm{PBKDF}_{(\mathrm{PRF},\, C)}\,(P, S, kLen).$$

The *kLen* value **shall** be at least 112 bits.

A generic diagram of the PBKDF is given in Figure 1. The design rationale on generic PBKDFs is available in Appendix A.2.

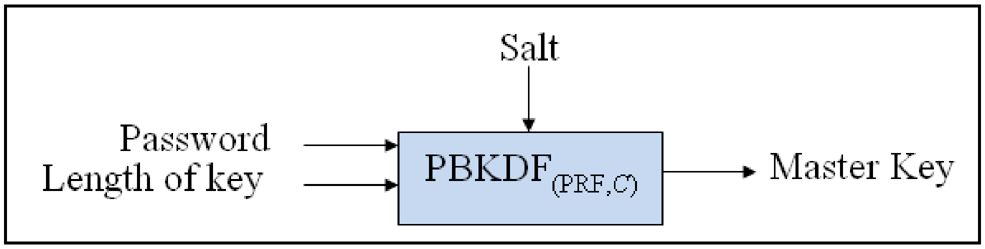


Figure 1: A generic diagram of the PBKDF

5.1 The Salt (*S*)

The salt **shall** be generated using an **approved** Random Bit Generator (e.g., see [6]). The length of the salt **shall** be at least 128 bits.

More information on the length of salt is available in Appendix A.2.1.

5.2 The Iteration Count (*C*)

The iteration count *C* is a fixed value that determines how many times the PRF iterates to generate one block of the MK. The iteration count **shall** be selected as large as possible, as long as the time required to generate the key using the entered password is acceptable for the users. Since the power of user machines varies (e.g., from a smart card to high-end workstations or servers), reasonable iteration counts vary accordingly. A minimum iteration count of 100,000 **shall** be used. For especially critical keys, or for very powerful systems or systems where user-perceived performance is not critical, an iteration count of 10,000,000 may be appropriate.

More information on the selection of the iteration count is available in Appendix A.2.2.

5.3 PBKDF Specification

The following algorithm for the derivation of MKs from passwords is based on an algorithm specified in [7], where it was specified as PBKDF2 and used HMAC [1] with SHA-1 as a PRF. This Recommendation approves PBKDF2 as the PBKDF using HMAC with any **approved** hash function as the PRF. The digest size of the hash function in bits is denoted as *hLen*. Figure 2 is a general diagram of the PBKDF.

The details of the PBKDF algorithm are given below.

```
Input:      P       Password
            S       Salt
            C       Iteration count
            kLen    Length of MK in bits; at most $(2^{32}-1) \times hLen$
Output:     mk      Master key
Algorithm:
    If $(kLen > (2^{32}-1) \times hLen)$
            Return “Desired key length is too long.” and stop ;
    $len = \lceil kLen / hLen \rceil$ ;
    $r = kLen - (len - 1) \times hLen$ ;
    For $i = 1$ to $len$
        $T_i = 0$;
        $U_0 = S \,||\, \mathrm{Int}(i)$;
        For $j = 1$ to $C$
            $U_j = HMAC(P, U_{j-1})$
            $T_i = T_i \oplus U_j$
    Return  $mk = T_1 \,||\, T_2 \,||\, \ldots \,||\, T_{len}\langle 0 \ldots r-1 \rangle$
```

5.4 Using the Derived Master Key to Protect Data

In this section, two options will be provided to protect data using a master key. In both options, a DPK is used to protect electronically-stored data, and the correctness of the MK **shall** be verified.

Note that if the DPK is compromised, it is necessary to recover (e.g., decrypt) the data and re-protect (e.g., re-encrypt) that data using a new DPK in both options.

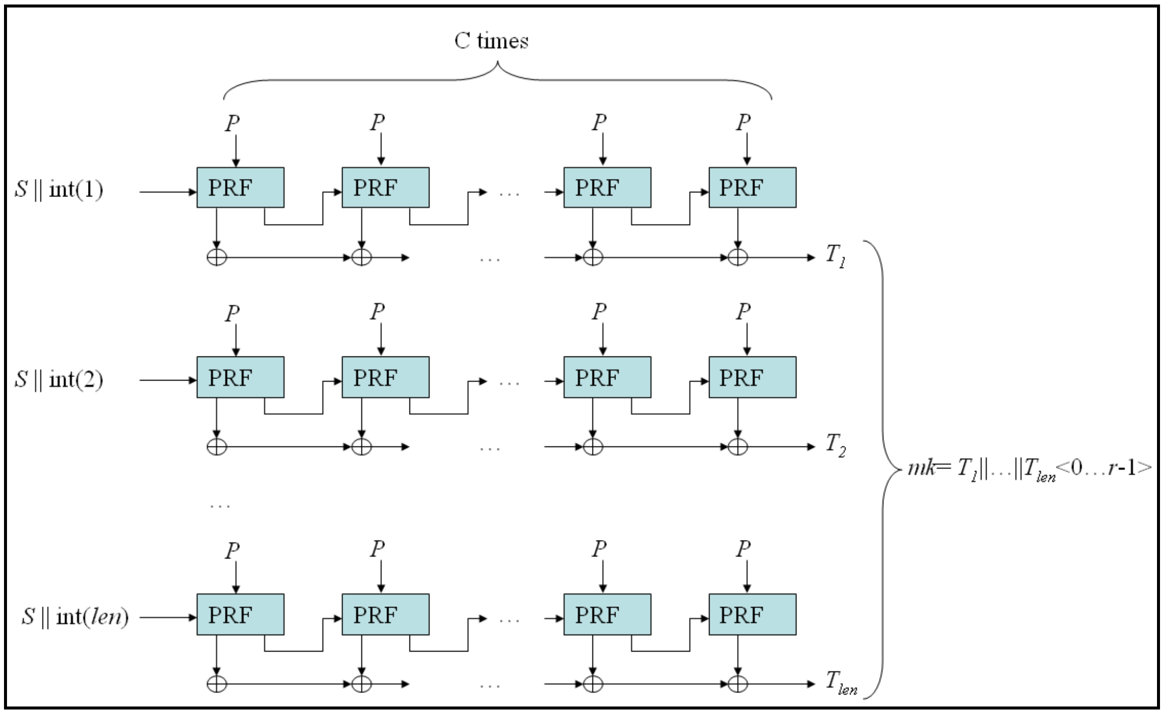


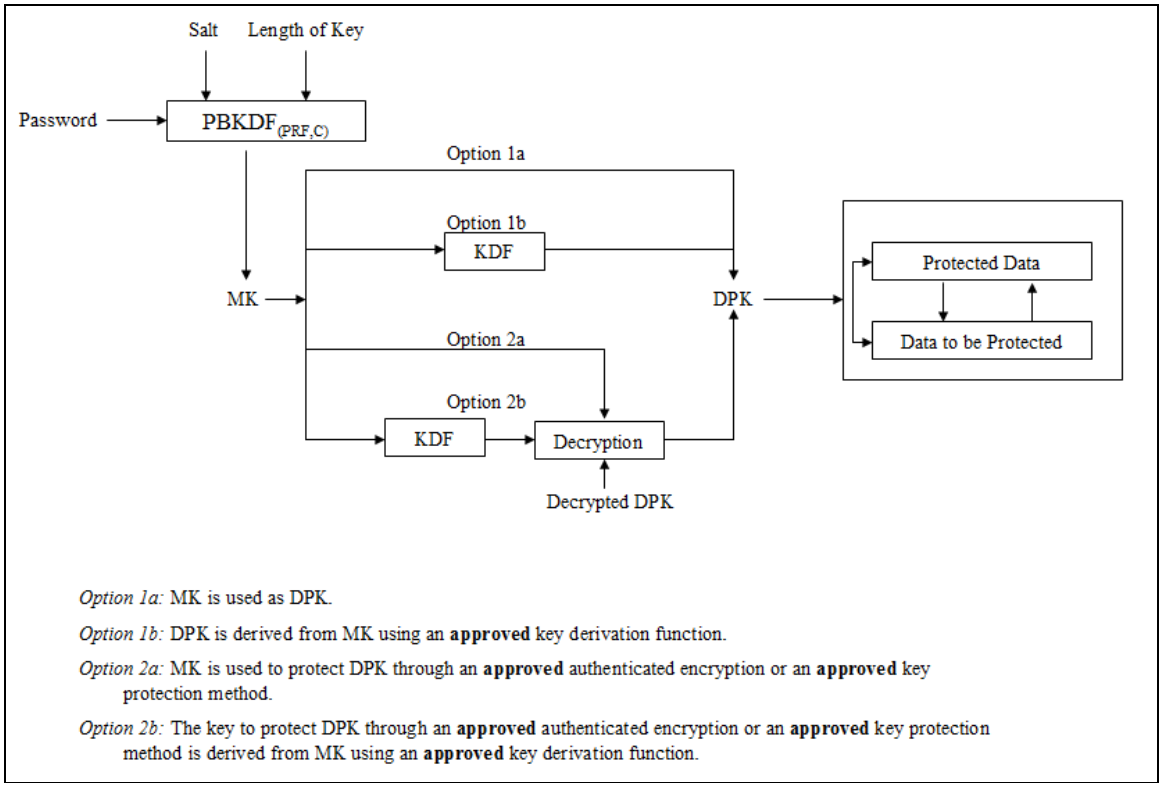
Figure 2: A general diagram of the PBKDF

Figure 3 summarizes a general procedure for password-based key derivation to access protected data (or apply protection to the data) with the two options.

Option 1:

In this option, there are two ways to derive DPKs from MKs: in Option 1a, the MK is used directly as the DPK, whereas in Option 1b, the DPK is derived from the MK using a KDF. Note that the DPK may be a set of keys used for encryption and integrity.

For options 1a and 1b, if only encryption is applied to the plaintext data, and the data size is large, in order to detect an incorrect entry of the password, the plaintext data might include some redundancy that can be checked easily without decrypting the whole data on the storage medium.



Option 1a: MK is used as DPK.

Option 1b: DPK is derived from MK using an **approved** key derivation function.

Option 2a: MK is used to protect DPK through an **approved** authenticated encryption or an **approved** key protection method.

Option 2b: The key to protect DPK through an **approved** authenticated encryption or an **approved** key protection method is derived from MK using an **approved** key derivation function.

Figure 3: A procedure for key derivation to access protected data or apply protection to the data.

This option requires that the entire protected data content be recovered (e.g., decrypted) and then re-protected (e.g., re-encrypted) whenever the password is changed.

Option 2:

In the second option, randomly generated DPKs are protected (e.g., encrypted) in two different ways. In Option 2a, an MK (or portions of an MK) that is generated from the password is used directly to protect the DPK. In Option 2b, an MK (or portions of an MK) that is generated from the password is used to derive keying material using an **approved** key derivation function, and the derived keying material (or portions of the derived keying material) is used to protect the DPK.

In this option, the protection of DPK **shall** use an **approved** authenticated encryption mode, such as defined in [3-5]. The MK is used to protect the DPK and to recover and/or verify the protected data.

The use of an **approved** authenticated encryption mode or key protection method allows the detection of an incorrect MK or incorrect derived keying material and, by extension, an incorrect password, thus avoiding the lengthy process of decrypting the protected data using an incorrect DPK. Depending on the authenticated encryption mode or key protection method used, data in addition to the salt and counter (e.g., an initialization vector) may need to be made available for decryption and encryption purposes (see Appendix A.3).

Whenever the user wants to change the password, a new MK computed using the new password is used to (re-) protect the DPK (either directly or indirectly, depending on the option), and the (re-) protected DPK is stored; this process does not require recovering and re-protecting the data on the storage media.

6 References

[1] FIPS 198-1, The Keyed-Hash Message Authentication Code (HMAC), July 2008

[2] NIST SP 800-108, Recommendation for Key Derivation Using Pseudorandom Functions, October 2009.

[3] NIST SP 800-38 C, Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality, May 2004.

[4] NIST SP 800-38 D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, November 2007.

[5] NIST SP 800-38 F, Draft Recommendation for Block Cipher Modes of Operation" Key Wrap Mode, expected December 2010.

[6] NIST SP 800-90, Recommendation for Random Number Generation Using Deterministic Random Bit Generators, March 2007.

[7] IETF RFC 2898 "PKCS #5: Password-based Cryptography Specification Version 2.0, September 2000.

Appendix A Security Considerations

A.1 User-Selected Passwords

A password is a secret string of characters that is used to gain access to a restricted resource. When passwords are used to generate cryptographic keys, it is assumed that an attacker is able to perform an offline attack on the key derivation process using a system of his own choosing, which may be much more powerful than the system used by the authorized user to generate the key from the password. Moreover, such attacks are always parallelizable. Therefore, the amount of entropy (roughly speaking, the number of guesses an attacker requires, on average, to determine a password) in the password is critical.

Randomly-generated passwords generally have much higher entropy than user-chosen passwords of the same length. However, in many applications, passwords are chosen by the users, because it is very hard to memorize randomly-generated passwords. For the security of electronically stored data, passwords **should** be strong enough so that it is infeasible for attackers to gain access by guessing the password. The strength of a password is related to its length and its randomness properties. Passwords shorter than 10 characters are usually considered to be weak. It is not advisable to use sequences of numbers or sequences of letters as passwords. Easily accessed personal information, such as the user's name, phone number, and date of birth, **should not** be included in a password.

A.2 Selection of the PBKDF

Dictionary attacks aim to recover passwords by trying the most-likely strings, such as words in a dictionary. These attacks are less likely to succeed against passwords that include random combinations of upper/lowercase letters and numeric values. Such passwords can only be recovered using inefficient *brute force attacks* that try all possible passwords.

The main idea of a PBKDF is to slow dictionary or brute force attacks on the passwords by increasing the time needed to test each password. An attacker with a list of likely passwords can evaluate the PBKDF using the known iteration counter and the salt. Since an attacker has to spend a significant amount of time for each try, it becomes harder to apply the dictionary or brute force attacks.

The output length of a PBKDF **shall** be at least 112 bits in order to thwart a brute force attack on the cryptographic key that is the output of the PBKDF.

A.2.1 Length of the Salt

The purpose of the salt is to allow the generation of a large set of keys corresponding to each password, for a fixed iteration count.

To avoid any possible interaction between other applications that use a salt, the salt may consist of two bit strings: a random value *rv* and the *purpose*, with the following form

$S = purpose \parallel rv$.

Purpose is the part of a salt that is reserved for an application-, message- or user-specific variable. The use of *purpose* is optional.

For a given password, the number of possible resulting keys is approximately 2^{sLen}, where *sLen* is the length of the salt in bits. For this Recommendation, the randomly generated part of the salt (*rv*) **shall** be at least 128 bits in length. Therefore, using a salt of at least this length makes it difficult for the attacker to generate a table of resulting keys, for even a small subset of the most-likely passwords.

A.2.2 Iteration Count

The purpose of the iteration count *C* is to increase the amount of computation needed to derive a key from a password, significantly increasing the workload of dictionary attacks. Using a PBKDF that requires *C* iterations to derive a key increases the computational cost of performing a dictionary attack on a password with *t* bits of entropy from 2^t

operations to $C \times 2^t$ operations, and therefore makes dictionary and brute force attacks more difficult.

However, the computation required for key derivation by legitimate users also increases with the number of iterations. The user will perceive this as an increase in the time required for authentication, or as an increase in the time to access the protected data on the storage medium. There is an obvious tradeoff: larger iteration counts make attacks more costly, but hurt performance for the authorized user. The number of iterations should be set as high as can be tolerated for the environment, while maintaining acceptable performance.

Acceptable performance is a function of the speed of the system doing the key derivation and the application. For example, consider a “whole disk” drive encryption application that boots a laptop computer from an encrypted drive, a fairly common application for password key derivation. The overall cold boot process, from the time the laptop is turned on until the system is loaded and available for use, may require thirty seconds to two minutes, and involve an operating system-level user authentication, as well as the entry of the boot password. A cold boot of a system is a relatively infrequent event. In such an application, it would be reasonable to set an iteration count that required ten seconds on the target machine, because the delay introduced would be small compared to the overall boot time.

At the other extreme, assume that a key is to be created as required for specific transactions, and that a user may do this many times an hour. The time to open the file independently of the password entry and key derivation is only a second or two. In this case, a delay of more than a second for the derivation of the key from the password will be apparent and annoying to the user. This suggests an iteration count that adds no more than a second on the target machine.

If the key derivation is to be performed on a server, then performance may be harder to gauge, and may depend on the peak load of the server. In many cases, however, even on servers, key derivation is only done once for a long session, and in these circumstances, a user-apparent delay of several seconds for key derivation may be tolerable.

A.3 Protection of DPK

Authenticated encryption schemes and key protection methods aim to protect both the authenticity and secrecy of a DPK. In Option 2 (see Section 5.4), a DPK is protected using an authenticated encryption scheme or an **approved** key protection method. If the user enters an incorrect password, it is impossible to generate the correct MK. However, using an incorrect MK results in a failure during the authentication process used in recovering DPK.

The additional variables needed to recover a DPK using the mechanisms specified in [3-5] (e.g., initialization vectors) may be stored on the hard drive or another storage device.